# Computation of Sectional Loads from Surface Triangulation and Flow Data

Shishir Pandya[*] and William Chan[†]

*NASA Ames Research Center, Mail Stop 258-2, Moffett Field, CA 94035*

**Traditionally, span-wise load distribution on a wing or the axial load distribution on a rocket is computed using approximations that utilize a combination of theory and empirical observations. The present effort describes a method with which sectional loads can be efficiently computed along an arbitrary direction from a computational fluid dynamics simulation. Efficient algorithms are presented so that a database of hundreds of solutions can be processed and plots that a designer can use can be generated within minutes. Software to compute the sectional loads is discussed. A user interface is provided to makes it easier to enter inputs and generate plots. The resulting software is scriptable for processing a database containing hundreds of solutions. As a byproduct of the method, cuts are produced and can be extracted to plot additional flow information along a cut, such as the pressure coefficient as a function of the chord distance.**

## I. Introduction

DURING the design of an aircraft, the designer must examine structural stress and strain on a wing, stall characteristics, aircraft performance, and the control response of a vehicle. All of these aspects of airplane design require the designer to have knowledge of the span-wise loading on the wing. Similarly, a rocket designer needs to know the load distribution along the main axis of the rocket for structural analysis. Historically, these sectional loads are computed using lifting-line theory[1–4] for wings. For missiles and rockets, axisymmetric body aerodynamics is combined with experimental and flight data to obtain sectional loads[5,6]. However, the use of computational fluid dynamics (CFD) in the design process is becoming common-place making it possible to predict sectional loads from a CFD simulation result.

The surface mesh and a flow solution on that surface can be extracted from a CFD simulation and this data can be used to compute the sectional loads. Most CFD simulations are performed using hexahedral or tetrahedral volume meshes resulting in surface data that is usually available as quadrilaterals or triangles. Today, the use of overset grid technology[7] is also common-place for simulation of complex bodies or for bodies moving relative to each other. For overset simulations, the surface is made up of a set of overlapping meshes. Though conventionally overlapping meshes are quadrilateral surface meshes, triangulated surface meshes or a mix are also possible. For all these mesh types, it is possible to convert the surface data to triangles. For structured meshes this can be done by simply cutting the quadrilaterals into triangles. For overlapping quadrilateral surface meshes, the MIXSUR code[8,9] of the Chimera Grid Tools (CGT)[10] package can take the overlapping surface data and produce a hybrid mesh with non-overlapping quadrilaterals and triangles. The quadrilaterals in the mesh can be subdivided to obtain a surface triangulation that contains the proper flow information. Thus, the flow data at each vertex of the surface triangulation is an appropriate starting point for the computation of sectional loads.

To compute sectional loads, the triangulation is binned geometrically normal to the sectional profile. New vertices are introduced where the bin boundaries intersect the triangle edges and flow data is interpolated to the newly created vertices. New triangles are formed using these vertices resulting in clear bin boundary definition. Loads are integrated inside each bin to obtain sectional loads. These sectional loads are then plotted to aid the designer in his/her analysis. Furthermore, the edges along each bin boundary are extracted during the process of introducing new triangles. These edges on each bin boundary are reordered to form a loop of connected edges. These loops are plotted along with the flow data on the vertices of these edges to obtain plots such as coefficient of pressure along that airfoil cut of the wing (e.g. $C_p$ vs. $x$) that are useful for aerodynamic analysis.

The CFD solution process in the design environment consists of creating a database of solutions to guide the design decisions. These databases cover a range of Mach numbers, angles of attack, side-slip angles, control surface deflections, and other changes in geometry. Thus, a database can become quite large incorporating hundreds of simulation results. The need for a large number of simulations forces us to consider the automation of post-processing the

[*]Aerospace Engineer, AIAA Senior Member
[†]Computer Scientist, AIAA Senior Member

solutions. A slow process needing a lot of user input can become quite cumbersome when the number of simulations is large, so an efficient process that requires little user input is desired. The present paper provides an efficient method to compute the sectional loads that can process a given simulation mesh and result in a few seconds, making it possible to post-process large databases in a few minutes. It also provides a method that is targeted to the user's needs asking for only the pertinent input, most of which is geared toward what the engineer needs to plot.

This paper focuses on describing an efficient method to compute and plot sectional loads and flow variables along cuts. Specifically, the first part of the paper discusses the details of the steps required to cut a given surface triangulation into bins (sections) according to user specification. The process of creating and using new vertices along bin boundaries to form triangles that define the distinct bins is described. The integration of the forces and moments in each bin is also discussed. Furthermore, the procedure that requires the use of vertices on the bin boundaries to create the boundary loops to make plots along the cuts (e.g. $C_p$ vs. $x$) is presented. Next, the development of a user interface to control the user options and to plot the results is also discussed. Finally, the method is validated on a flat plate and demonstrated on a wing, and a rocket through a presentation of the resulting sectional and cumulative loads.

## II. Approach

The results from a CFD simulation of fluid flow around a vehicle body is assumed to be the starting point for computing sectional loads. These simulations provide the flow data at each vertex of the volume mesh from which the flow data on the surface mesh can be extracted. As discussed earlier, we choose to start from a surface triangulation and flow data at every point in the triangulation because any surface representation can be cut into triangles including overlapping quadrilateral patches. Furthermore, the triangle is a simplex that can support data in a bi-linear representation and thus linear interpolation of flow data onto newly introduced vertices at bin boundaries is sufficiently accurate making a triangulation an ideal choice.

To compute the sectional loads from this triangulation, the triangulation must be chopped into sections called bins in a user-specified direction. New points are introduced where the edges of the input triangulation are cut by a bin boundary and the flow data is interpolated to these points. The triangles that straddle the boundary are broken up into multiple triangles so that the forces and moments in each bin can be uniquely and independently integrated. These sectional loads are added up to obtain the cumulative loads. Because the sectional loads depend on the size of the bin, the same surface and flow data will yield a different sectional load for a different number of bins. A derivative of the cumulative loads can also be computed using finite differences to represent a sectional load that is independent of the section size. The process is generalized for any number of bins and any axis orientation. Often the input surface is broken up into components (e.g. fuselage, wing, vertical tail, etc.). These component definitions can also be arranged in groups to obtain sectional loads on a subset of the vehicle. Furthermore, the force and moments can be divided into contributions from pressure, viscous, and momentum. These can also be separately reported to enable more specific analysis of the resulting loads.

Each step of the process shown in Fig. 1 is discussed in detail in this section. An efficient method for cutting a triangulation with arbitrary planes is described along with the algorithm for bin separation and computation of forces and moments. A cost estimate for the process is also provided. These methods are implemented in a code called TRILOAD as a tool in the CGT package.

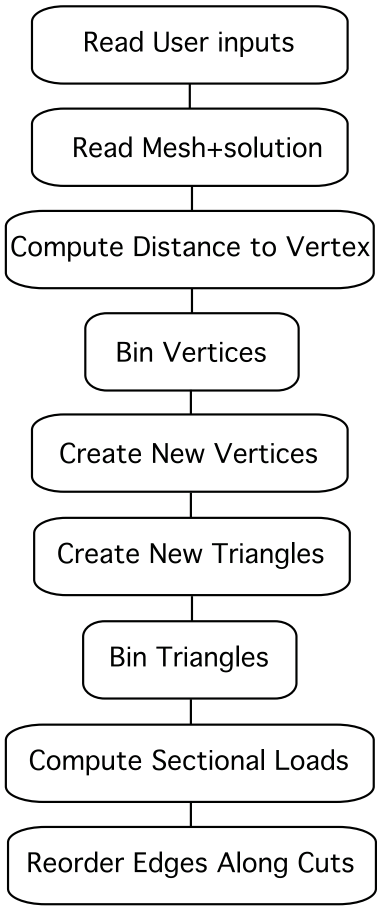


**Figure 1. The sectional loads computation process.**

### A. Distance to vertex from minimum plane

In addition to providing the surface triangulation and flow data, the user must also provide some guidance for the cuts. At a minimum, the number of bins, bounds for the cutting domain ($S_{min}$ and $S_{max}$) and direction of cuts are required to determine how the cuts should be made. These parameters can be used to determine if an existing triangle edge is cut by one or more of the bin boundaries (shown in green in Fig. 2) by determining which bin each existing vertex lies in. The most efficient method of assigning bin numbers to vertices is to use the distance, $D$, between the vertex and the minimum-plane. For the three vertices of a triangle, the distances

$D_1$, $D_2$, and $D_3$ are depicted in Fig. 2. Using the origin of the geometry's coordinate system as the reference, the vertex distance is simply the inner product of the unit vector that defines the axis along which the bins are cut with the point in question.

This axis vector can be specified by the user in three ways. For cuts aligned with one of the Cartesian planes, the axis vector is simply $(1, 0, 0)$, $(0, 1, 0)$ or $(0, 0, 1)$ for constant $x$, $y$ and $z$ cuts respectively. For an analytically defined plane $ax + by + cz = constant$, the normal to the plane is the gradient of the function (i.e. $a\hat{i} + b\hat{j} + c\hat{k}$). Therefore, the unit axis vector is the unit vector associated with the vector $(a, b, c)$. Finally, one can specify three points that define a plane so that each cut is parallel to this plane. In this case, the axis vector can be computed by the cross product of the vector $P_1P_2$ with the vector $P_1P_3$ where $P_1$, $P_2$, and $P_3$ are the three points in the plane.

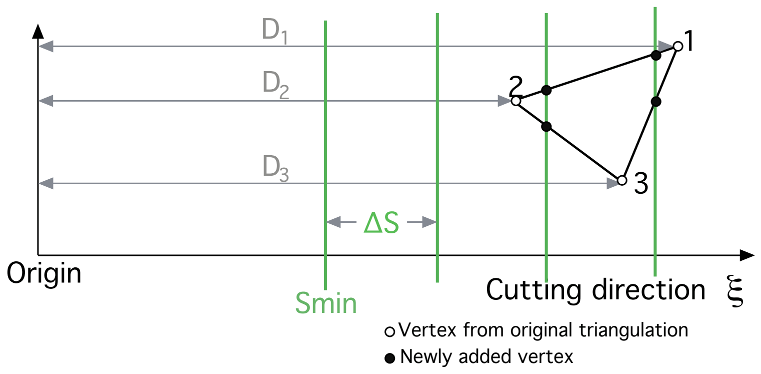


**Figure 2. An edge cut by bin boundaries.**

## B. Placing vertices in bins

Using the distance function, one can identify which bin the vertex is in using the bin size

$$\Delta s = \frac{S_{max} - S_{min}}{N_b} \tag{1}$$

where $S_{min}$ and $S_{max}$ define the domain size and $N_b$ is the number of bins. The domain size can either be user-controlled, or can be associated with the bounding box of the geometry. From the bin size, the vertex bin number can be calculated as $bin_v = integer(\frac{D_v}{\Delta s}) + 1$.

## C. Estimation of additional vertices

Once the existing vertices are assigned a bin number, it is now possible to make a conservative estimate of the number of vertices and triangles that will need to be added to obtain perfectly defined slices of the geometry. Such an estimate is necessary to properly allocate memory for the sliced up triangulation.

Two options are available for adding vertices to cut the triangulation; triangle traversal and edge traversal. However, they result in different triangulations and are not equally efficient. Due to efficiency considerations for large databases of simulations, we need to choose the most efficient method. The efficiency is determined by the number of operations required to obtain the new triangulation. The triangle traversal method requires us to go to each existing triangle and see if its edges are cut by a bin boundary. However, two neighboring triangles share an edge meaning that each operation to determine if an edge is cut must be done twice. Furthermore, the introduction of the new vertices as well as the interpolation of the flow variables to these new vertices are all duplicated operations for each newly introduced vertex. The edge traversal method avoids this duplication and is thus more efficient.

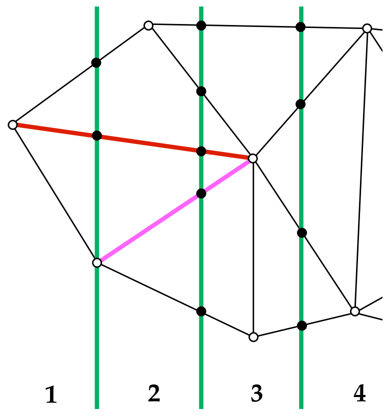


**Figure 3. An edge (red) cut by bin boundaries (green).**

With edge traversal, the number of new vertices can be counted by simply visiting each existing edge of the triangulation. At each edge, the minimum and maximum bin numbers are obtained from the vertex bin assignment. Since the edge is made of only two vertices, the minimum and maximum bin numbers are easily obtained. For example, Fig. 3 shows an edge of the triangulation in red for which $maxbin = 3$ and $minbin = 1$. This edge is cut by two bin boundaries shown in green so that the new triangulation will have two new vertices. Thus, the number of new vertices along an edge is defined as $maxbin - minbin$. Adding up contributions from all edges results in the total number of vertices that need to be added.

## D. Introducing new vertices on bin boundaries

The creation of new vertices is similar to counting the possible vertices in that the procedure is done edge by edge. As discussed during the counting procedure, we already know how many times the edge will be cut. So a loop from $minbin + 1$ to $maxbin$ is setup to obtain the cuts. Inside the loop, the location where the bin boundary cuts the edge needs to be found. The process for this is depicted in Fig. 4 where we assume that the edge (shown in red) varies from $\xi = 0$ to $\xi = 1$. Using the distance function, and normalizing by $D_{v2} - D_{v1}$ achieves this goal such that the vertex $v_1$ is at $\xi = 0$ and $v_2$ is at $\xi = 1$. The cutting plane distance for the $j$th bin boundary is also known by use of $\Delta s$ as

$$D_p = S_{min} + (j-1) * \Delta s \tag{2}$$

Now the cutting plane (shown in Fig. 4 in green) location can be computed as

$$\xi_{cut} = \frac{D_p - D_{v1}}{D_{v2} - D_{v1}} \tag{3}$$

The location of the new point can now be calculated as

$$\vec{X}_{cut} = \vec{X}_{v1} + \xi_{cut}(\vec{X}_{v2} - \vec{X}_{v1}) \tag{4}$$

Similarly, $\xi_{cut}$ can also be used to interpolate flow variables to the new point.

The index of the first vertex added to an edge is stored (in an array called EtoV) so that it can be used later to break up triangles. Since all additional vertices are introduced along an edge consecutively and the number of added vertices is known, only the index of the first vertex needs to be stored.

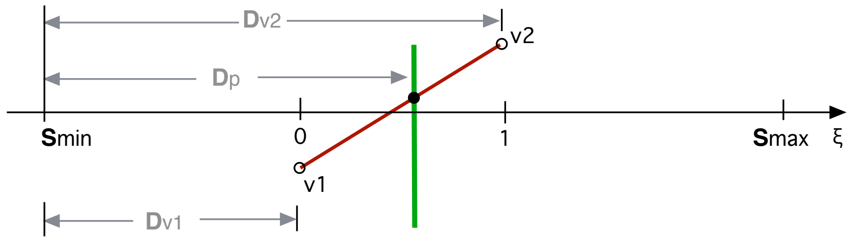


**Figure 4. Computing the location of the cutting plane along an edge.**

While introducing new vertices, if $\xi_{cut}$ is identically zero or one, then a degeneracy exists where one of the two vertices lies on a bin boundary. Such vertices can be treated as internal to later obtain zer-zrea triangles. However, in an effort to create an algorithm that considers all possible degeneracies, these vertices are flagged so that they can be accounted for when creating new triangles.

## E. Estimation of additional triangles

Before creating new triangles, an upper bound can be established for the number of triangles that need to be added. Here again, efficiency considerations are needed. The triangle traversal is used, but two options are available for cutting the existing triangles. One method is to introduce a new triangle every time a new vertex is introduced. This results in a simple algorithm, but is not the most efficient because for each new vertex introduced in an original triangle, a new triangle results. This means that a newly introduced triangle may need to be cut into more triangles if more than one bin boundary cuts that triangle. Thus, this method results in an operations count that goes up exponentially with the number of bin boundaries that cut an original triangle.

A method with a predictable operations count must, therefore, take into account all newly introduced vertices on the edges of an existing triangle. Thus, new vertices are introduced along each edge and stored in memory so that they can be recalled and used when cutting a triangle. With this method, a triangle is cut into either two or three triangles when only one bin boundary cuts it. If the bin boundary cuts two edges of the triangle, three triangles result(see magenta cuts in Fig.5). If only one edge is cut and the bin boundary passes through an existing vertex, two triangles result (see red cuts in Fig. 5). If instead of one, two bin boundaries go through a triangle as depicted in blue in Fig. 5, then it can be cut into a maximum of five triangles. In a degenerate case (depicted in cyan in Fig. 5) where one of the vertices rests on a bin boundary, the number is reduced to four triangles. A conservative estimate is obtained by assuming that there are no degenerate vertices. Thus, for each existing triangle the maximum number of new triangles is $(maxbin - minbin) * 2$ where the min and max bins are for the three vertices of the triangle. For example, the vertex farthest to the right in both blue and cyan triangles is in bin 3, and the one farthest left is in bin 1. Thus, $maxbin = 3$ and $minbin = 1$. So the number of new triangles is counted as $(3-1) * 2 = 4$. For the blue triangle which is cut into five triangles, one triangle replaces the original and four new triangles are introduced.

## F. Creation of new triangles

Triangles are by definition convex geometric objects. Thus, tessellation of a triangle that has been cut by a plane is a trivial task. Thus, rather than treat this as a general triangulation problem, we use the knowledge that all triangles are cut by bin boundaries that are parallel to each other. This consideration allows us to examine every possible triangle creation scenario. These scenarios are treated in three parts: minmum bin, intermediate bins, and maximum bin. To determine if an existing triangle needs to be cut into 2 or more triangles, we examine the stored information for the three edges of the triangle. If none of the edges have a proper vertex assignment, the triangle does not need to be split up. If a proper vertex assignment is found, the triangle is split up starting with the minimum bin and working our way to the maximum bin. For all possible triangulation scenarios, all possible degeneracies are considered and treated such that every possible variation is accounted for in the resulting algorithm. Note that this process is presented generically, and must be triplicated for each of the three possible vertex configurations.

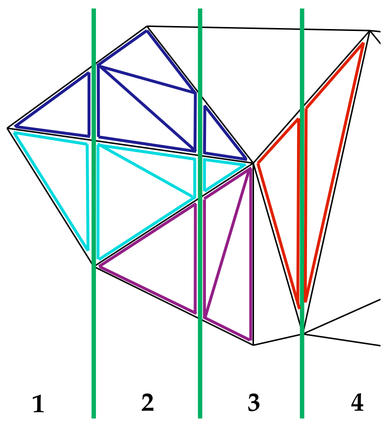


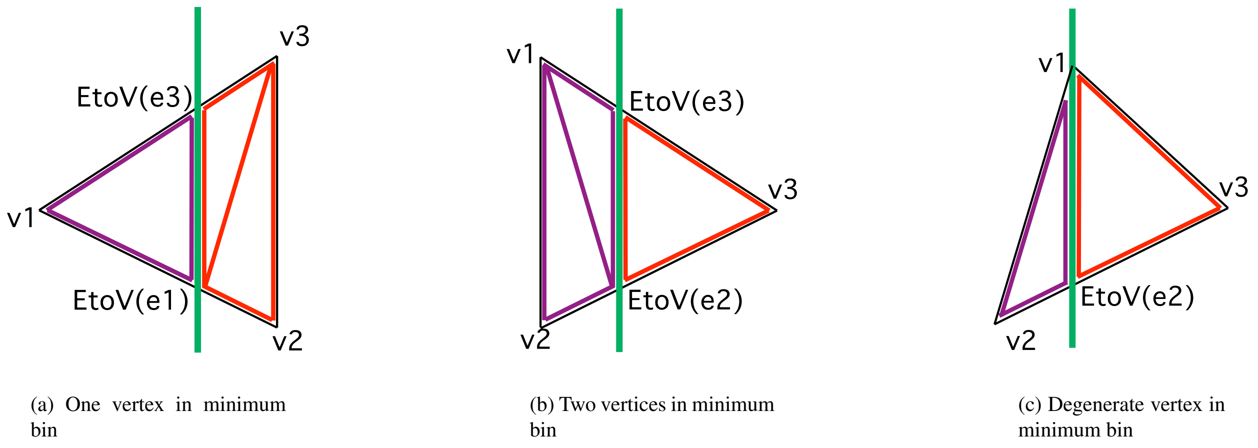
**Figure 5. A triangle cut by bin boundaries.**

### *1. Minimum bin*

In the minimum bin, there are several possible ways in which a bin boundary may cut through the triangle. We divide these into three categories; when one vertex is in the minimum bin, when two vertices are in the minimum bin, and when there is a degeneracy such as a bin boundary passing through a vertex of the original triangle.

**One vertex in the minimum bin:**

The first possibility is that only one of the vertices of the existing triangle is in the minimum bin (see magenta triangle splits in Fig. 6(a)). In this case, we save the vertex information from the original triangle in a temporary space and replace the existing triangle with the triangle.



(a) One vertex in minimum bin

(b) Two vertices in minimum bin

(c) Degenerate vertex in minimum bin

**Figure 6. Minimum and maximum bin triangle cuts.**

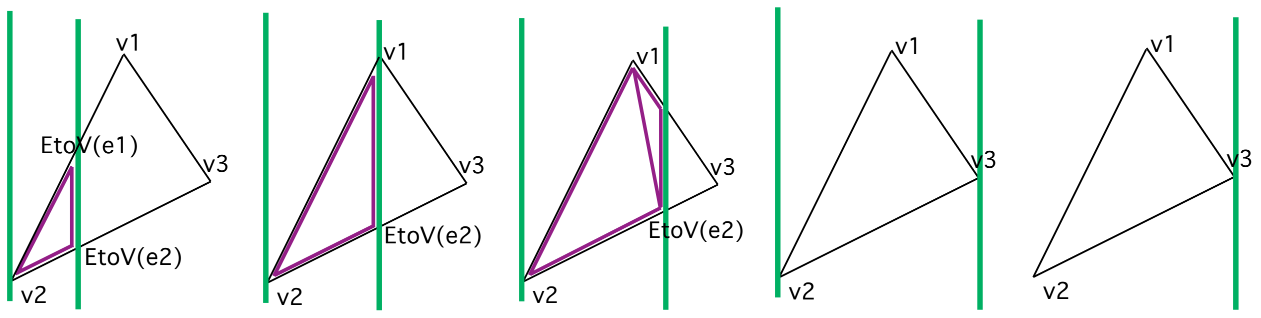
**Two vertices in the minimum bin:**

The second possibility is that two of the three vertices of the triangle are in the minimum bin (see Fig.6(b)). In this case, the part of the triangle in that bin is cut into two triangles. One of those triangles replaces the original triangle, while the other is added to the end of the triangle list.

**Degenerate possibilities:**

In the case of a degeneracy, one or more of the vertices of the triangle will lie on a bin boundary. Remembering that bin numbers are assigned to vertices such that a vertex on a bin boundary is assigned to the bin on the right of the boundary, five possibilities arise. The first possibility is that of a vertex on the minimum bin boundary and a second bin boundary going through the triangle(see Fig. 7(a)). In this case, the original triangle is replaced with the new triangle which consists of the vertex on the $minbin$ boundary and the two new vertices introduced by the second bin boundary.

The second case is similar to the first, but the second bin boundary goes through another vertex of the triangle(see Fig. 7(b)). The original triangle in this scenario is replaced with a triangle connecting the two original vertices with the new vertex on the edge that is cut by the second bin boundary. In the third case, the bin boundary has moved father to the right such that one of the vertices is in the minimum bin with the third vertex in a higher bin (see Fig. 7(c)). This case results in two triangles. One replaces the original and one new triangle is introduced. A fourth possibility is that the second bin boundary has moved right so that it now goes through the third vertex(see Fig. 7(d)). Here, none of the edges have been cut by a bin boundary and so the triangle does not need to be sub-divided. The last possibility is depicted in Fig. 6(e). Here only the right-most vertex of the triangle touches the bin boundary. Again, the triangle does not need to be sub-divided. Note that in cases one through three, which vertex is on the boundary must be determined in order to assure proper definition of the new triangles (with a normal that matches the original triangle).



(a) One vertex on the $minbin$ boundary

(b) Previous case with a vertex on an intermediate bin boundary

(c) Case (a) with a vertex in the $minbin$

(d) A vertex on $minbin$ and $minbin + 1$ boundaries

(e) A vertex on the $minbin$ boundary

**Figure 7. Minimum and maximum bin triangle cuts.**

## 2. *Intermediate bins*

The possible triangle break-ups in an intermediate bin can be categorized the same way as the minimum bin.

**One vertex in the minimum bin:**

The case of a single vertex in the minimum bin is depicted in Figs. 8(a) and (b). Both cases have a vertex in the minimum bin with two or more bin boundaries passing through the triangle. The first case has two bin boundaries passing through the edges attached to the vertex in the minimum bin. This results in two new triangles in the intermediate bin. The second case has one bin boundary cutting through the same two edges and another on that cuts one of those two edges again and the third edge of the triangulation. This means that one of the original vertices of the triangulation is in the intermediate bin and thus, three triangles result in this bin instead of two.

**Two vertices in the minimum bin:**

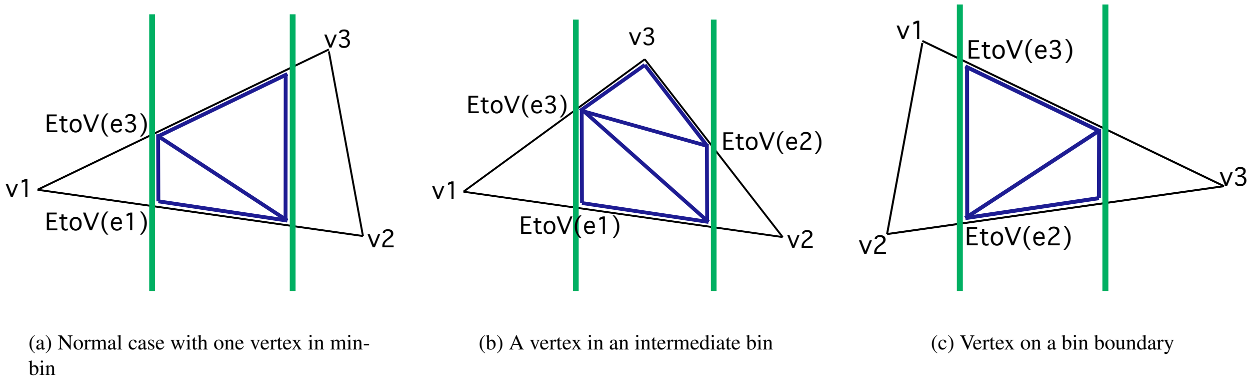
The third case, depicted in Fig. 8(c), is opposite of the case in (a) in that two vertices are in the minimum bin. However, it results in a similar set of two triangles since two of the triangles edges get cut by two bin boundaries.

For all cases, the vertex assignments of the new triangles are dependent on the EtoV array which holds the vertex number of the first vertex that cuts an edge. To obtain the vertex numbers at each subsequent cut along an edge, we simply add one to the EtoV array.

**Degenerate possibilities:**

Several degenerate cases occur when a bin boundary passes through a triangle vertex and are shown in Figs. 9. The two cases depicted in Fig. 9(a) are similar. The first is where a vertex is on the high bin boundary of the intermediate bin and the second is where the same vertex is on the low bin boundary of the intermediate bin. Both result in two new triangles, but the details of the triangles differ and proper attention must be paid to make sure that the new triangles have a proper normal.

Figures 9(b-d) portray a set of cases where a bin boundary passes through the right-most vertex. All these cases are treated as intermediate bin cases because the right-most vertex is assigned the next bin number. The simple way to distinguish between these cases is that if there is an original vertex in the intermediate bin, two new triangles must be introduced. However, if the same original vertex is coincident with a bin boundary or cuts through the triangle without



(a) Normal case with one vertex in min-bin

(b) A vertex in an intermediate bin

(c) Vertex on a bin boundary

**Figure 8. Intermediate bin triangle cuts.**

interaction with any of the original vertices (as in Fig. 9(d), then only one new triangle is introduced. Finally, a case where a bin boundary passes through two of the original vertices on the right-most side of the triangle is shown in Fig. 9(e) and results in two new triangles.

**Algorithmic details:**

The flow of the triangle construction is as follows. First a loop over all possible bins ($minbin + 1$ to $maxbin - 1$) assures that new triangles are introduced in each bin. Within the loop, we first determine which one of the four cases discussed above must be presently addressed. This is done by checking if a vertex is in the current bin, or on a bin boundary. Since each vertex is assigned a bin number, and the vertices on a bin boundary are already flagged when bin numbers were assigned to vertices, this is a trivial task. We now turn our attention to the actual assigning of triangles. Starting with the left most bin, we determine which two edges are active. Note that only two edges can be cut by a bin boundary, and thus two edges must be active unless a degeneracy exists. Once the active edges are known, triangles are created using the EtoV array entries associated with those edges. Active edge assignments must be modified when a vertex is in the bin since the presence of a vertex in a bin indicates that an edge has ended in the bin. The EtoV array entry associated with the newly activated edge must be used to construct the triangles in the present bin as well as subsequent bins. In the case where a vertex is on a bin boundary, the active edge assignment must also be modified. However, this situation presents two opportunities to make this modification. We choose to make the modification when the degenerate vertex is on the left bin boundary (or at the last opportunity).

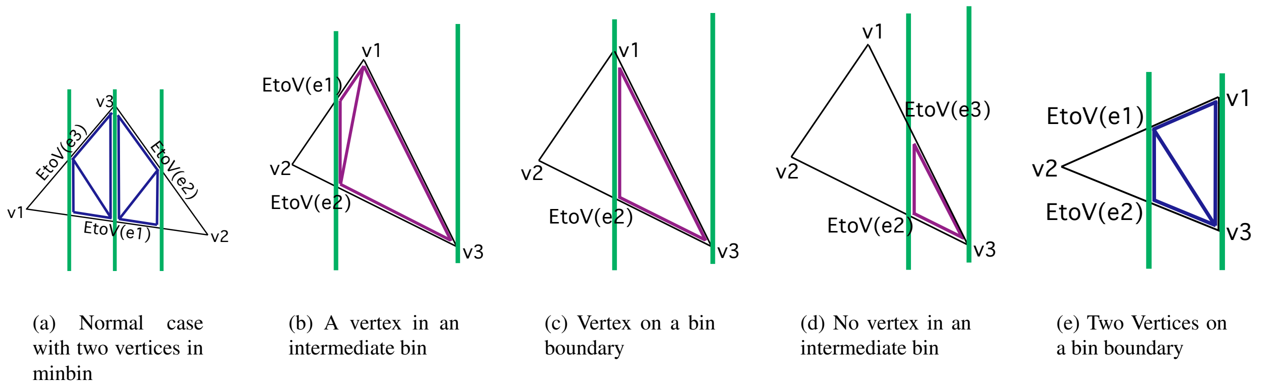
Finally, each of the case discussed above has two possibilities in its own right. For example, if edges 1 and 3 are active in the left most intermediate bin and a vertex is in the present bin (as depicted in Fig. 8(b)), either vertex 3 (v3) or vertex 2 (v2) could be in the bin. Each of these two cases results in a different set of vertex assignments to the newly introduced triangles. Similar possibilities also exist when the vertex is on a bin boundary. Also, the treatment of a normal case is different when the normal case occurs after an inactive edge was activated due to the presence of a vertex in a bin or on a bin boundary. The same vertices are used to assign new triangles, but the same order will result in opposite facing normals. Thus, the vertex order must be changed in these cases. In all, there are twenty three possible ways in which bin boundaries can cut through a triangle. For each of these types, the three vertex permutations discussed above must be accounted for.

### 3. *Maximum bin*

This part of the problem is very similar to the minimum bin problem(see red triangle splits in Fig. 6) since the same two possibilities (one or two vertices in maximum bin) exist here with two exceptions. The first exception is that whereas one of the triangles in the minimum bin replaces the original triangle, all triangles in the maximum bin are added triangles. The other difference arises in the handling of the degenerate case. If only one vertex is in the maximum bin and it is on the bin boundary, then no triangle needs to be added. However, if two vertices are in the maximum bin and one of them is on the bin boundary, only one triangle needs to be added. Finally, if both vertices in the maximum bin are on the bin boundary, no triangles need to be added.

## G. Reordering edges along cuts

During the process of adding triangles, edges from new triangles that lie on a bin boundary are flagged. The set of edges on each bin boundary can then be used to plot $C_p$ vs. $x$ if the edges are reordered to be a contiguous set. To



(a) Normal case with two vertices in minbin
(b) A vertex in an intermediate bin
(c) Vertex on a bin boundary
(d) No vertex in an intermediate bin
(e) Two Vertices on a bin boundary

**Figure 9. Intermediate bin triangle cuts.**

reorder the edges, we start with the first edge in the list of edges on each bin boundary. The list of all edges (on the current bin boundary) is now traversed to find an edge that shares a vertex with this edge. Because the vertices are globally numbered, the search is only an integer comparison to other vertex indices avoiding the need to compute distances between vertices. Once the edge loop is complete, the locations of the vertices associated with the edge loop can be written out with the flow data so that plots such as $C_p$ vs. $x$ can be made.

## H. Computation of the forces and moments

In addition to the triangulation information, the input file must also hold flow information. Twelve scalars are required to compute pressure, viscous, and momentum forces and moments. They are $\rho$, $u$, $v$, $w$, $p$, $\mu$, $u_\zeta$, $v_\zeta$, $w_\zeta$, $\Delta x = x_{l+1} - x_l$, $\Delta y = y_{l+1} - y_l$, $\Delta z = z_{l+1} - z_l$. Here, $\rho$ is the density, $u, v, w$ are the three Cartesian components of velocity, $p$ is the pressure, and $\mu$ is the viscosity. The next three variables define the gradient of velocity with respect to the wall-normal direction and the last three variables hold the location of the point above the surface so that a prism volume can be computed associated with each triangle.

The computation of forces from these scalars can be divided into three parts: Pressure force, viscous force, and momentum force. These are discussed in the subsections below.

### *1. Pressure force*

To compute the forces due to pressure, only the $C_p$ is needed. Since

$$C_p = \frac{P - P_\infty}{\frac{1}{2}\rho_\infty u_\infty^2} \tag{5}$$

and pressure is normalized with

$$\bar{P} = \frac{P}{\rho_\infty a_\infty^2} \tag{6}$$

$C_p$ becomes

$$C_p = \frac{\bar{P} - \bar{P}_\infty}{\frac{1}{2}M_\infty^2} \tag{7}$$

Now, the force due to pressure can be computed using

$$\vec{F}_p = \int -P d\vec{A} \tag{8}$$

However, it is often computed for closed bodies using

$$\vec{F}_p = \int -(P - P_\infty) d\vec{A} \tag{9}$$

In the present framework, the user can decide which of these option he/she wants to use.

Now the coefficient of pressure is defined using a reference area $A$ as follows

$$\vec{C}_{F_p} = \frac{F_p}{\frac{1}{2}\rho_\infty u_\infty^2 A} \quad (10)$$

$$= \frac{\int -(\bar{P} - \bar{P}_\infty) d\vec{A}}{\frac{1}{2}M_\infty^2 A} \quad (11)$$

$$= \int -C_p d\vec{A} \quad (12)$$

To obtain the dimensional value of force, we compute

$$\vec{F}_p = \bar{F}_p \frac{1}{2}\rho_\infty a_\infty^2 L^2 \quad (13)$$

$$= \int -(\frac{1}{2}M_\infty^2 C_p dA)\frac{1}{2}\rho_\infty a_\infty^2 L^2 \quad (14)$$

where the unit grid length $L$ must be specified by the user in meters for SI and ft for English units. The user must also specify $\rho_\infty$ in $Kg/m^3$ for SI and in $Slugs/ft^3$ for English units. Finally, the value of the speed of sound $a_\infty$ must be specified in $m/s$ for SI and $ft/s$ for English units.

### 2. *Viscous force*

The viscous force is computed from the viscous stress tensor on a triangle as

$$F_i = T_{ij} A_j \quad (15)$$

where $T_{ij}$ is the stress tensor and $A_j$ is the vector area of the current triangle. The stress tensor on a triangle is computed from the average $\vec{u}_\zeta$ on the triangle. The diagonal terms of the tensor are

$$T_{ii} = \frac{2\mu}{ReV}(u_{\zeta_i} A_i - \frac{1}{3}\vec{u}_\zeta \cdot \vec{A}) \quad (16)$$

and the off-diagonal terms are

$$T_{ij} = \frac{\mu}{ReV}(u_{\zeta_j} A_i + u_{\zeta_i} A_j) \quad (17)$$

where $Re$ is the Reynolds number, and $V$ is the Volume of the prism associated with the triangle.

### 3. *Momentum force*

The momentum force needs to be calculated if there is a boundary with flux across it such as an inlet or an exhaust from an engine. For these situations, the flux across a triangle is computed as

$$Flux = \vec{u} \cdot \vec{A} \quad (18)$$

The average flux is computed using the average velocity vector on the triangle. The force can now be computed as

$$\vec{F}_m = -Flux(\rho\vec{u}) \quad (19)$$

or

$$\vec{F_{mi}} = -\int (\rho u_i u_j n_j) dA \quad (20)$$

### 4. *Moments*

The moments can simply be derived as

$$\vec{M} = \vec{r} \times \vec{F} \quad (21)$$

The only additional consideration is that the moment must be multiplied by an additional unit grid length ($L$) for dimensionalization and divided by reference length in the non-dimensional computation.

### I. Accuracy and Load Conservation

All data is required to be stored at the vertices of the triangulation. This means that the data is represented in a bilinear manner across each triangle or linearly along each edge of the triangulation. Higher order representations are possible with introduction of intermediate vertices, but are not considered here. Thus, when the process of cutting the triangulation into bins introduces new vertices along the edges of the existing triangulation, the data is interpolated linearly to the new vertex from the two vertices of the original edge. The resulting data representation is identical to the original representation because the triangle is a simplex and cutting it into multiple triangles does not change the representation of the data. Thus, the resulting algorithm is load-conserving. Because the newly created triangles are completely nested in the original triangulation, the integration is also moment-conserving. This load-conservation property is demonstrated in the flat plate subsection of the results section.

### J. Computational cost

It is possible to do the same task with many different algorithms. For example, instead of creating the new vertices and then using them to create new triangles, one can do both of those operations simultaneously. This results in an algorithm where an existing triangle is split into 2 or 3 triangles every time a bin boundary cuts it. The problem with this algorithm is that the newly created triangle now also has to be cut into 2 or 3 triangles when another bin boundary is found to go through it. The number of operations for large initial triangles can thus mount exponentially as a function of the number of bins. The present algorithm avoids this problem by introducing new vertices at every bin boundary on each original edge cut by the boundary. Thus, the introduction of new vertices as well as triangles scales linearly as a function of the number of bins. The cost of each operation to achieve the goal of computing forces and moments can be estimated by examining the loop structure for the operation. For each task, the number of operations scale linearly and are presented in Table 1.

**Table 1. Cost of operation. $N$ is the number of vertices, and $N_b$ is the number of bins.**

| Function | Number of operations scaling |
|---|---|
| Distance computation | $\mathcal{O}(N)$ |
| Vertex binning | $\mathcal{O}(N)$ |
| New vertex estimate | $3\mathcal{O}(N)$ |
| New triangle estimate | $2\mathcal{O}(N)$ |
| New vertex creation | $3\mathcal{O}(N)\mathcal{O}(N_b+1)$ |
| New triangle creation | $2\mathcal{O}(NN_b)$ |
| Computation of force and moments | $2\mathcal{O}(N)$ |
| Scaling to obtain dimensional values | $\mathcal{O}(N_b)$ |
| Total | $11\mathcal{O}(N)+5\mathcal{O}(NN_b)$ |

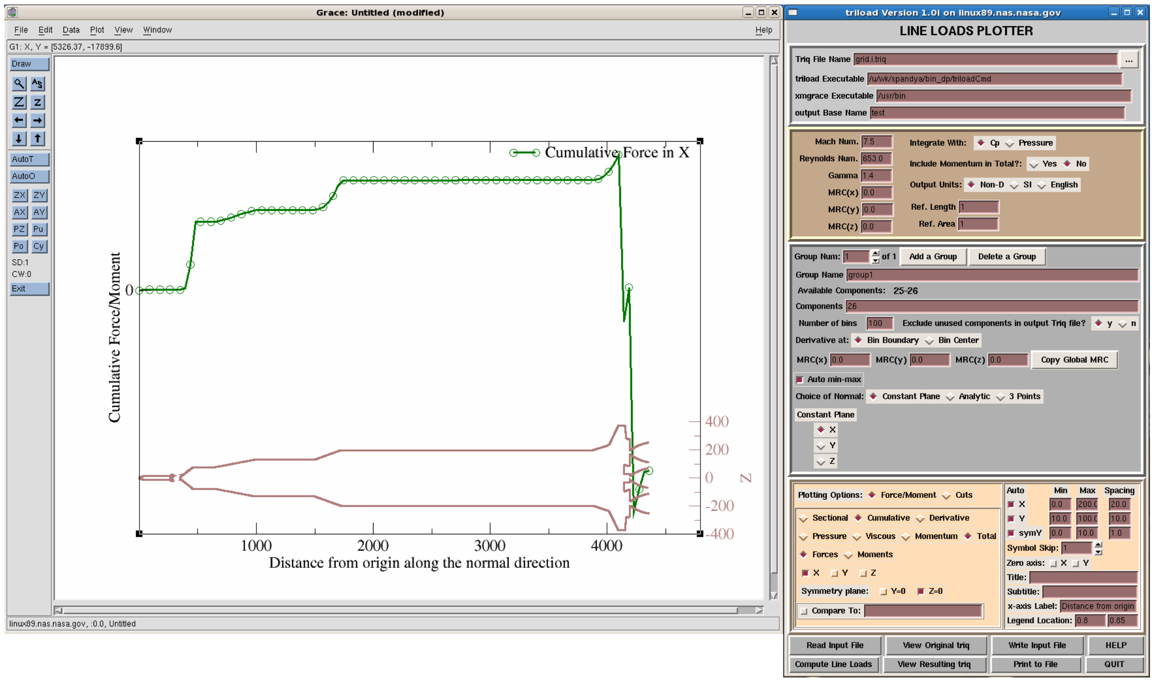
### K. Execution

The resulting software is called TRILOAD and can be used on the command line, in a graphical user interface (GUI), or from a script. The command-line makes it possible to test or debug the sectional loads computation for just one CFD solution. The GUI makes it possible to easily enter input and plot data for a single case, while the scripts allows the user to post-process a database containing hundreds of solutions in an efficient manner to obtain the plots that a design engineer can use to assess various aspects of the aircraft or rocket's aerodynamics, structure, and stability and control issues.

### L. Graphical user interface

A graphical user interface (GUI) is designed to make it easier for the user to input the user-specified options as well as to facilitate efficient interrogation of the results. Figure 10 shows the GUI consisting of two parts. The left side of the GUI is an xy-plotting window from the Grace package[11]. The right side of the window is the user input and plotting controls module written using Tcl/Tk[12,13]. The Tcl language is used to setup the variables and read and write input files as well as manage the user inputs. The Tk widget set is used for the interface so that the user can simply enter the necessary inputs or click on a button to choose an option or issue a command. Finally, the Grace software and the Tcl/Tk part of the GUI are interfaced using a pipe. The GUI sets up a pipe and initiates Grace so that it can read and execute commands from that pipe. Commands are then written to the pipe to initiate actions in Grace (e.g. read a data file, choose what to plot, control axes scales, input titles and subtitles). As the user clicks buttons or enters information in the Tk GUI on the right hand side, the Grace window on the left is updated and the appropriate xy-plot appears on the left hand side.

The right hand side window is broken up into five blocks. The top block allows the user to input the name of the input file, the locations of the executables and a base name for output files. The second block allows the user to input and update information about the input data file. This includes the Mach number, and Reynolds number as well as a global moment reference center. The same pane also allows the user to make choices such as whether the force/moment integration for pressure force should use pressure or $C_p$. The user can also control the output units here and input the appropriate values for reference variables for the chosen output units.



Triq File Name: grid.i.triq
triload Executable: /u/wk/spandya/bin_dp/triloadCmd
xmgrace Executable: /usr/bin
output Base Name: test

Mach Num.: 7.5
Reynolds Num.: 653.0
Gamma: 1.4
MRC(x): 0.0
MRC(y): 0.0
MRC(z): 0.0
Integrate With: [✓ Cp] Pressure
Include Momentum in Total?: Yes [✓ No]
Output Units: [✓ Non-D] SI English
Ref. Length: 1
Ref. Area: 1

Group Num: 1 of 1
Group Name: group1
Available Components: 25-26
Components: 26
Number of bins: 100
Exclude unused components in output Triq file? [✓ y] n
Derivative at: [✓ Bin Boundary] Bin Center
MRC(x): 0.0 MRC(y): 0.0 MRC(z): 0.0
[✓ Auto min-max]
Choice of Normal: [✓ Constant Plane] Analytic 3 Points
Constant Plane: [✓ X] Y Z

Plotting Options: [✓ Force/Moment] Cuts
Sectional [✓ Cumulative] Derivative
Pressure Viscous Momentum [✓ Total]
[✓ Forces] Moments
[✓ X] Y Z
Symmetry plane: Y=0 [✓ Z=0]
Compare To:

| Auto | Min | Max | Spacing |
|---|---|---|---|
| [✓ X] | 0.0 | 200.0 | 20.0 |
| [✓ Y] | 10.0 | 100.0 | 10.0 |
| [✓ symY] | 0.0 | 10.0 | 1.0 |

Symbol Skip: 1
Zero axis: X Y
Title:
Subtitle:
x-axis Label: Distance from origin
Legend Location: 0.8 0.85

**Figure 10. Graphical User Interface (GUI) showing x-y plotter to the left and input and control widgets on the right.**

The third block allows the user to control the size, and orientation of the cuts as well as to create and manage multiple groups. Each group is identified by a group name and defined by the component numbers that are included in the group. The user controls the bin size by specifying the number of bins and the extent of the cuts by specifying a min and max for each group. Finally, the user can define the cutting direction by specifying it as a constant plane, or as a plane made of three points. An analytical method of defining the plane as $ax + by + cz = constant$ is also provided.

The inputs from these three blocks are used to compute sectional loads and their values can also be saved to a file. The controls for these actions are provided in the bottom block along with the ability to examine the triangulation (both original and newly sliced) using the OVERGRID[14] program.

Once the sectional loads have been computed, the fourth block is used to plot the results in the xy-plotter to the left. The plotting options allow the user to change various aspects of the plot.

## III. Results from example test cases

The method is demonstrated on several example test cases and is verified for an exact solution. A geometry of a wing protruding from a box is used to demonstrate the cutting and assigning of bins. A falt plate is then used to validate the integration method. Following that, the Common Research Model (CRM) wing-body from the Drag Prediction Workshop (DPW)[15] is used to illustrate the force/moment computations. The same wing is also used to show the reordered cuts and the resulting $C_p$ vs. $x$ plots. Finally, the Saturn V rocket is used to demonstrate the capability on a rocket geometry. In all cases, the results are presented in non-dimensional units.

## A. A wing protruding from a box

Figure 11 shows a wing protruding from a box. This test geometry is used to show that the geometry can be cut in an arbitrary way. Three example options are used to show that the cuts can be made in a direction of the user's choosing and that the cuts can be limited by component numbers, or minimum and maximum specification. The input geometry has two components; the box and the wing. Fig. 11(a) shows the result of choosing to cut only the component containing the wing in the $x = constant$ direction. Colors are used to distinguish the bins and the eight bins on the wing can be seen clearly in this figure. Figure 11(b) shows the same body, but only the box is cut up. The direction of the cuts is specified at a 45° angle using the analytical plane method (e.g. $a = 1$, $b = 1$, $c = 0$ for $ax + by + cz = constant$). The maximum value is also set such that a part of the box does not get cut. Finally, Fig. 11(c) shows both the wing and the box cut along the constant z plane. Similarly, any orientation and number of bins can be specified along with a minimum and a maximum value to obtain the desired cuts. Furthermore, the algorithm does not prevent the user from specifying the exact locations of the cuts if such cutting location values are known a-priori. An output file containing the resulting triangulation is used to visually verify that the cuts obtained are the ones that the user requested.

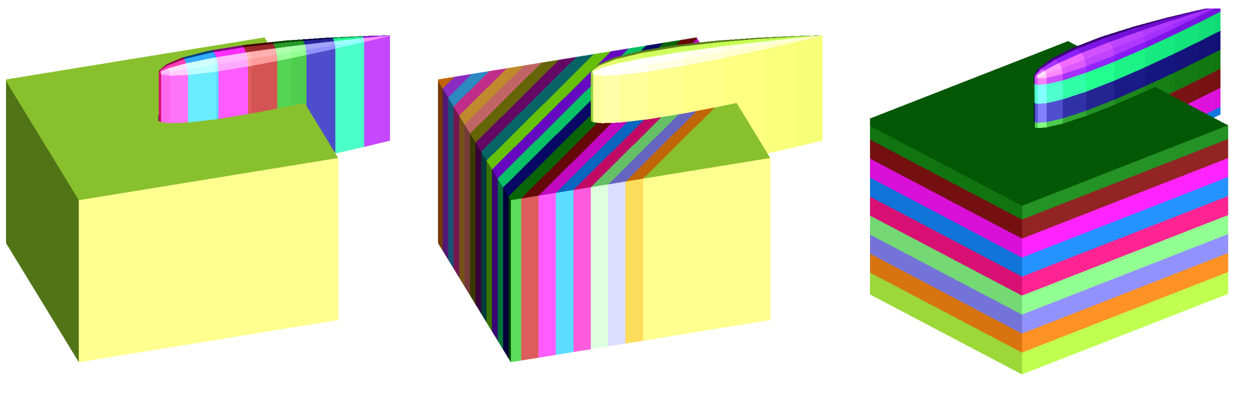

(a) Constant x cuts on the wing (b) 45° cuts on the box (c) Constant z cuts on both

**Figure 11. Wing-in-box examples of cutting a geometry.**

## B. A flat plate with exact solutions

A flat plate with a specified function, $f(x)$, is used to validate the integration procedure. To do this, a flat plate triangulation is created from a Cartesian mesh of 21x21 points (see Fig. 12). Analytically, the sectional loads are computed on a flat plate that goes from 0 to 1 in both $x$ and $y$ directions as follows.

$$C_z = \int -C_p dS = \int_0^1 \int_{section} f(x)dxdy \tag{22}$$

where $S$ is the area of the triangles being integrated in the bin. Here, at each vertex of the triangulation, the value of the specified function, $f(x)$ is loaded into the $C_p$ value such that $-C_p = f(x)$. The resulting triangulation with the $f(x)$ data at each vertex is written to a file and the file is fed into the sectional loads algorithm to obtain sectional and cumulative loads from cuts along the $x$ direction. These sectional loads are computed so that for a bin

$$C_z = \int_{section} f(x)dx \tag{23}$$

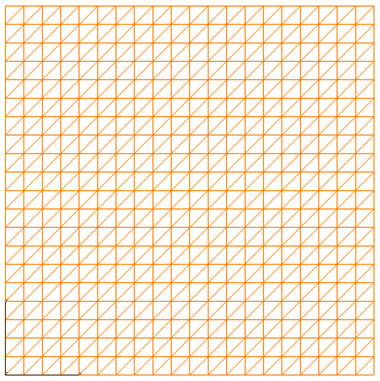


**Figure 12. A 21x21 mesh on a flat plate.**

### 1. Linear function

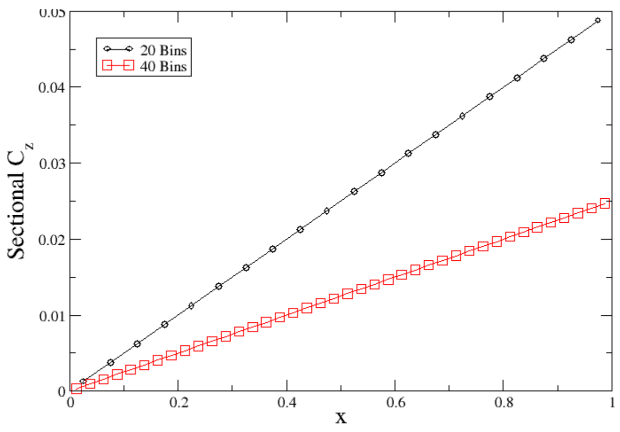
If $f(x) = x$, the sectional load should be

$$C_z = \int_0^1 x\Delta x dy = x\Delta x \tag{24}$$

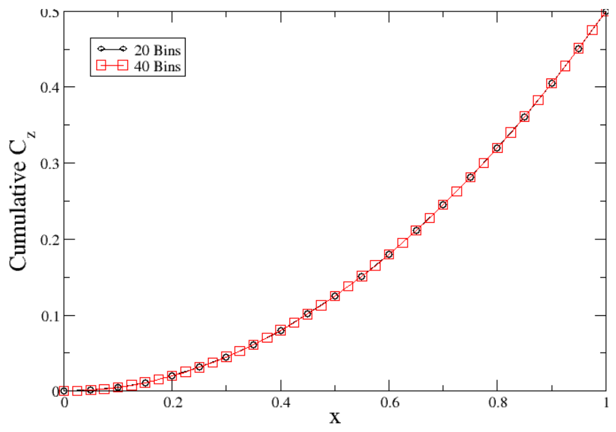
For 20 bins, $\Delta x = 0.05$, thus, the sectional force coefficient is expected to be $C_z = 0.05x$. Since $x$ varies from 0 to 1, $C_z$ must vary linearly from 0 to 0.05. For 40 bins, $\Delta x = 0.025$, thus, the sectional force coefficient is expected to be $C_z = 0.025x$ and varies linearly from 0 to 0.025. Integrating to obtain the cumulative loads

$$\int f(x)dx = \int x dx = \frac{1}{2}x^2 \tag{25}$$

The resulting quadratic function is, therefore, expected to vary between 0 and 0.5. The sectional and cumulative loads for the linear $f(x)$ are shown in Fig. 13, and match the expected results.



(a) Sectional load.

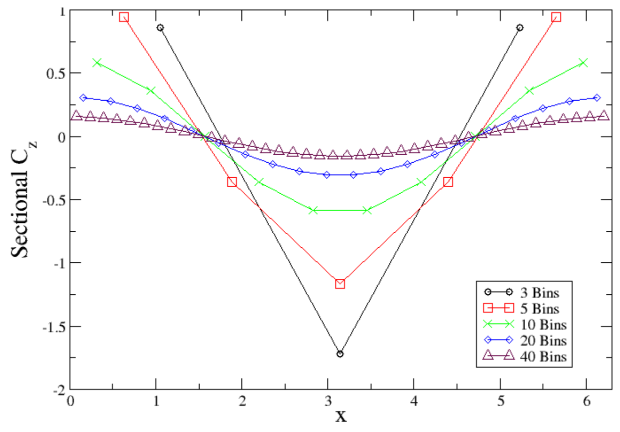


(b) Cumulative load.

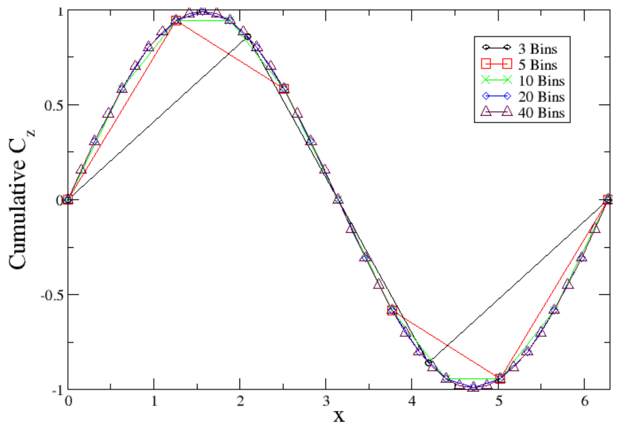
**Figure 13. Linear function on a flat plate with two different number of bins.**

## 2. *Trigonometric function*

Whereas a linear function can be represented on a mesh made of simplexes identically, there is an error associated with the representation of non-linear functions. Therefore, the integration of a transcendental function is investigated. The sectional and cumulative loads for $f(x) = cos(x)$ are shown in Fig. 14.



(a) Sectional load.



(b) Cumulative load.

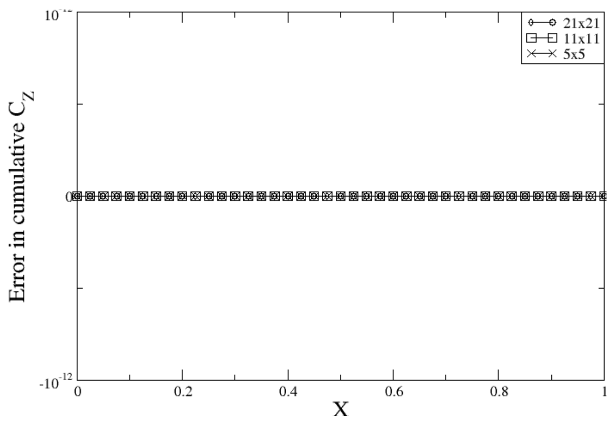
**Figure 14. Trigonometric function on a flat plate with 21x21 original mesh with various number of bins.**

For $f(x) = cos(x)$, the sectional load is predicted to be $cos(x)\Delta x$. Here, the domain is extended in the $x$ direction so that it ranges from 0 to $2\pi$ to assures that we capture a single cycle of the cosine wave. For 10 bins, $\Delta x = \frac{2\pi}{10} = \frac{\pi}{5}$, thus, the sectional force coefficient is $C_z = \frac{\pi}{5}cos(x)$. which must vary from $-\frac{\pi}{5}$ to $\frac{\pi}{5}$. As the number of bins is increased, the amplitude of the cosine wave should reduce and vise versa. Integrating to obtain the cumulative loads

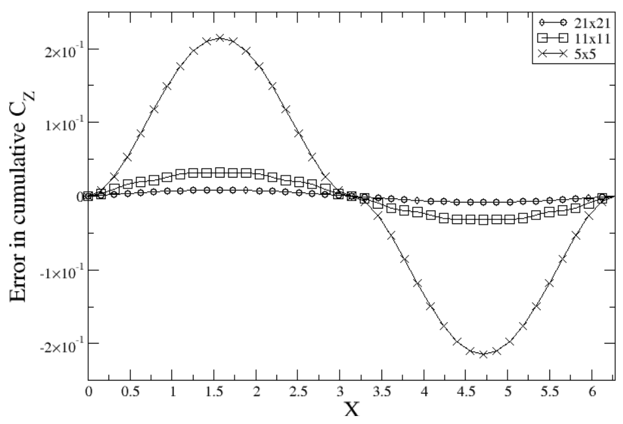
$$\int f(x)dx = \int cos(x)dx = sin(x) \tag{26}$$

This prediction matches the results shown in Fig. 14. This figure also illustrates the simplex property of a triangle in that it shows that the representation across the triangles does not change due to the introduction of new vertices. The figure shows results for the triangulation cut into 3, 5, 10, 20, and 40 bins. For each of these choices, the cumulative load lies as close to the exact solution as the original mesh can support. Note that all points on the cumulative load plot lie on the same curve regardless of the number of bins.

## 3. *Accuracy*



(a) Linear function.



(b) Trignometric function.

**Figure 15. Error in the cumulative load as a function of $x$ with various initial meshes.**

This example can also be used to verify that the expected accuracy and load conservation properties are attained. Because a mesh made of simplexes can support linear functions identically, the error in the computed load must be identically zero for linear functions. Conversely, a non-linear function can not be represented identically and the error in the solution can only be reduced by increasing the grid density. The exact solutions for the cumulative load for the linear and the cosine functions can be used to compute the error at every point. Figure 15 plots the error as a function of $x$ for three input meshes; a 5x5, an 11x11, and a 21x21 mesh. Figure 15(a) shows zero error for the linear function regardless of the initial mesh resolution, proving that the algorithm computes loads for linear functions identically.

Figure 15(b) shows the error for the cosine function for three meshes of the same resolution as those in 15(a). This figure shows that a coarse input mesh of 5x5 has a large error (about 20%) and as the meshes get finer, the error is reduced. Figure 16 shows a plot of the error as a function of initial mesh size. The error reduction is shown to be 2nd order. In other words, everytime the initial mesh density is doubled, the error in the integrated values goes down by a factor of 4.

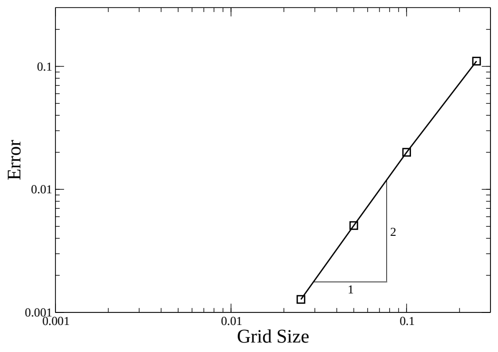


**Figure 16. Initial mesh refinement study.**

As most real life flow solutions are non-linear, the user must provide enough grid density to adequately represent the solution. However, the loads integration on the triangles is highly accurate as shown by the linear function. For non-linear functions, this can be further seen in Fig. 17 where the cumulative load is plotted for the cosine function on a 5x5 input mesh for various numbers of bins. The integrated cumulative loads under-predict the exact solution (shown as a solid black line) as discussed earlier, but the values all lie on the same curve regardless of the number of bins used to obtain sectional loads. This demonstrates that the integrated load has the same error regardless of the choice of bins and therefore the error is associated purely with the representation of the data on the input mesh.

## C. The CRM wing

The Common Research Model (CRM) wing geometry (Fig. 18(a)) from the drag prediction workshop[15] is used to demonstrate the sectional loads computations. The $M = 0.85$ with $Re = 18129$ case is run using the Overflow-2 code[16] and the surface meshes are extracted and converted to a triangulation using the MIXSUR tool[8]. The resulting surface is in three parts as seen in Fig. 18(a). The wing component (shown in green) is cutup into 20 bins along the y axis (spanwise) and the resulting triangulation is shown in Fig. 18(b). Each triangle here is colored by its bin number. The forces and moments are integrated in each bin. The lift force is shown as a function of the span-wise distance from the origin in Fig. 19. Fig. 19(a) shows the sectional force, while Fig. 19(b) shows the cumulative force. The sectional lift force is seen to be the highest near the root and dropping off to zero near the tip as expected.

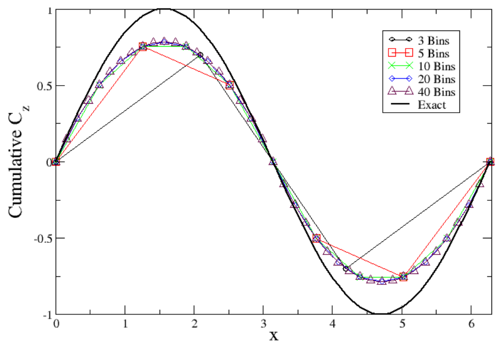
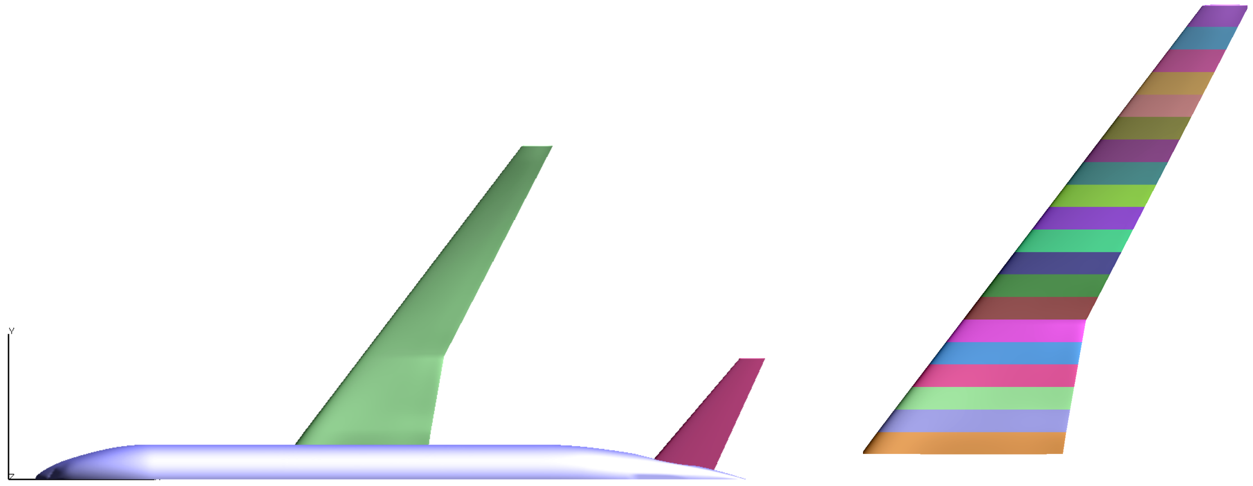


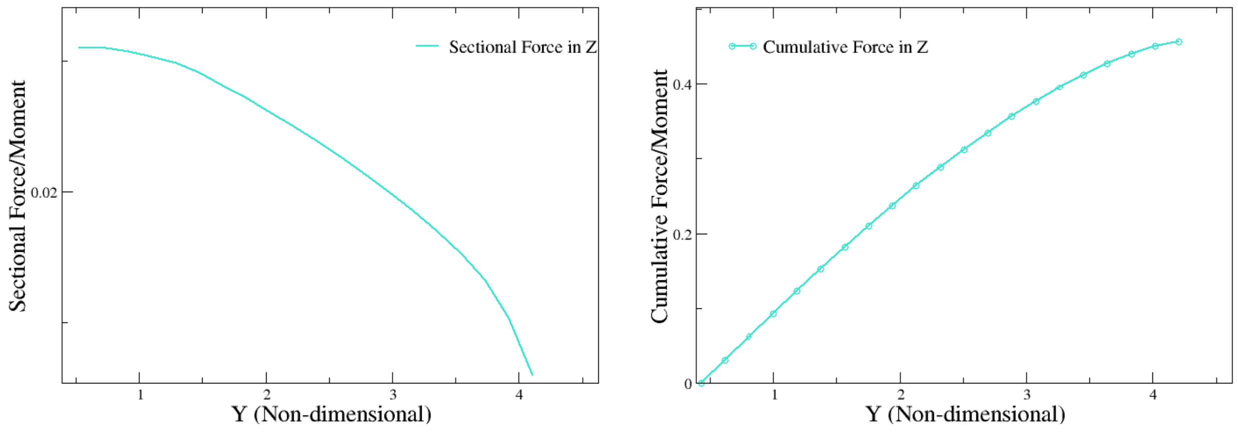
**Figure 17. Cumulative load for a cosine function on a 5x5 initial mesh using various number of bins.**



(a) Geometry.

(b) Wing triangulation cut into bins.

**Figure 18. The CRM wing-body.**



(a) Sectional.

(b) Cumulative.

**Figure 19. The span-wise lift forces on the CRM wing.**

The CRM wing-body's original triangulation has 300,000 triangles and the wing is made up of 135,000 of those. To obtain 20 bins and the associated cuts for $C_p$ vs. $x$, the algorithm takes one third of a second. The resulting triangulation of the wing has 160,000 triangles. Increasing the number of bins to 200 results in an increase in time to compute the results to 0.9 seconds. Increasing the number of bins to 2000, results in a computation time of 7.3 seconds.

The $C_p$ vs. $x$ cuts extraction capability is also illustrated on the same geometry. Three bin boundaries, root, mid-section and outboard section, are shown along with the $C_p$ at those cuts in Fig. 20. The capability to plot cuts as well as to compare them to external data (e.g.experimental results) is also available from the GUI.

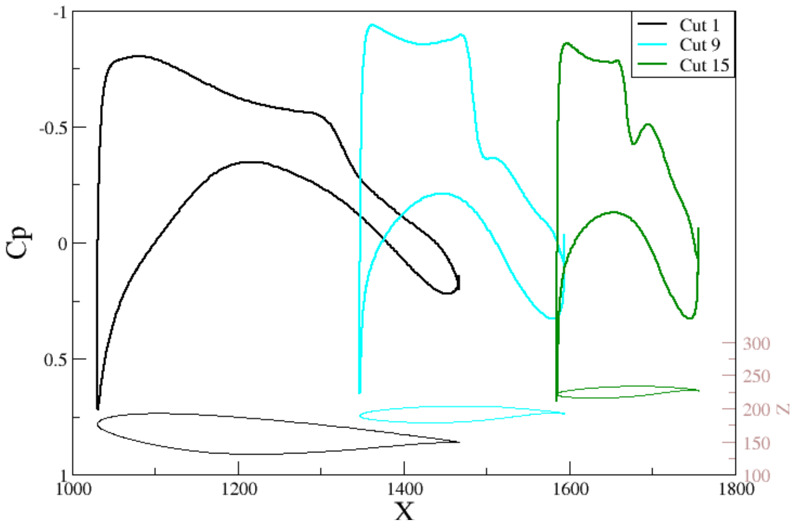


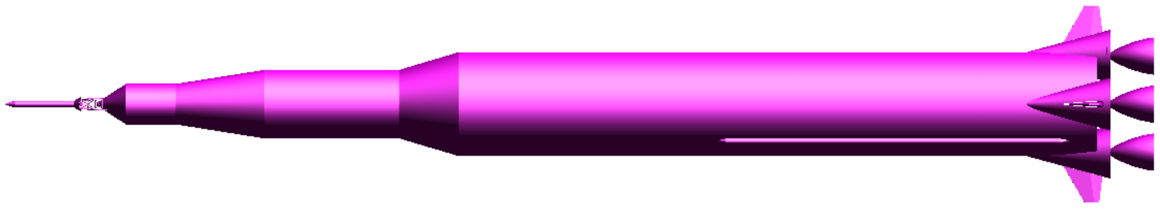
**Figure 20. $C_p$ vs. $x$ for the CRM wing at three bin boundaries.**

## D. The Saturn V rocket

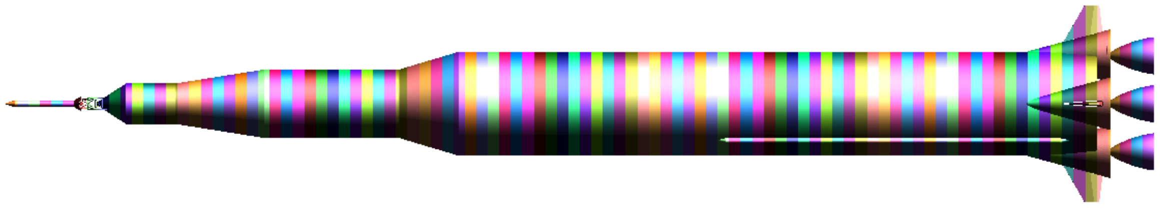
The Saturn V rocket is used to demonstrate the sectional loads capability for a rocket. The geometry of the Saturn V rocket is shown in Fig. 21(a). The sectional loads are computed with cuts in the $x$ (axial) direction, shown in Fig. 21(b), using one hundred bins. A simulation of the rocket at $M = 7.5$ is performed using the Overflow-2 Code[16] with plumes from the F-1 rocket engines. The result is post-processed using the MIXSUR code from CGT to obtain a surface triangulation and associated flow data. The current algorithm is then used to compute sectional loads in each bin. Fig. 22(a) shows the sectional axial force (drag) from the simulation. The figure shows that the large changes in the drag force along the axial direction correspond to the changes in the diameter of the rocket body. In the rear, where the fins, flares, and nozzles are, substantial loads are observed. This translates into the cumulative loads shown in Fig. 22(b). This figure shows that each increase in the diameter of the rocket is an increase in the drag force which is the expected behavior. The lift and side forces are shown in Fig. 23(a) and show that these forces are negligible on the fore-body which is mostly axi-symmetric. On the aft-body, the systems tunnels, as well as the flares, fins, and nozzles contribute to substantial changes in the lift and side forces. To see these effects in the moments, all three moments (pitch, roll, yaw) are plotted in Fig. 23(b). The moments show the same trend in that they are nominally zero on the fore-body. On the aft-body, the systems tunnels and the flares and fins cause a change in the pitch and yaw while the largest jump is observed due to the nozzle area in the pitching moment.

## IV. Concluding remarks

An efficient algorithm for computing sectional loads from a CFD solution is developed. A triangulation on the surface of the body and associated flow data are used as input. It is shown that cutting the body triangulation into sections along an axis of interest (e.g. span-wise) and integrating the forces in each of the sections is an effective method of computing the sectional loads. As the bin boundaries are available, the edges along this boundary are extracted and reordered so that they can be used to plot $C_p$ vs. $x$.

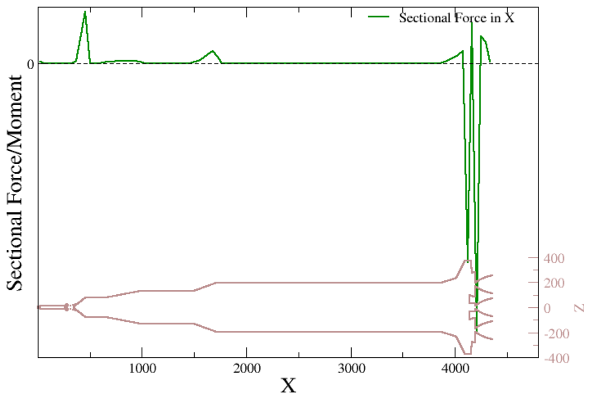


(a) Geometry.

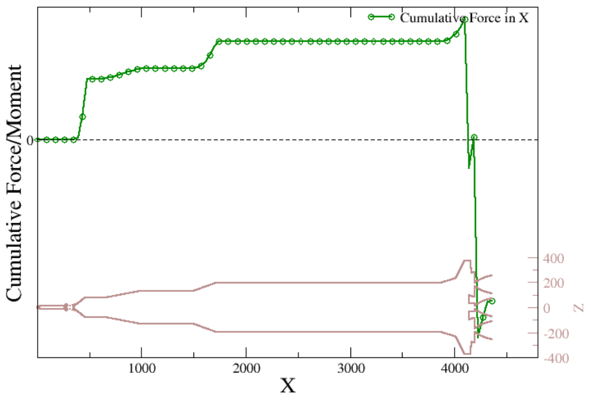


(b) Rocket triangulation cut into bins.

**Figure 21. The Saturn V rocket.**

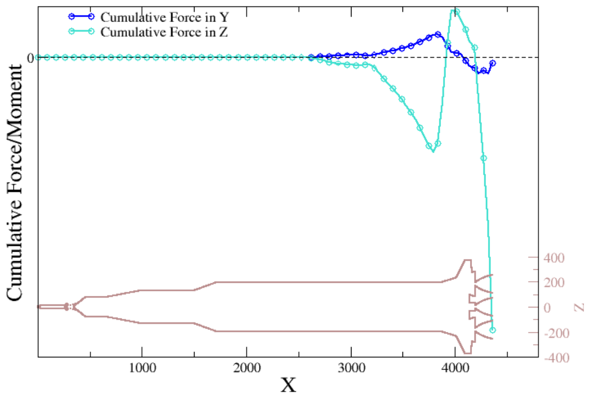


(a) Sectional.

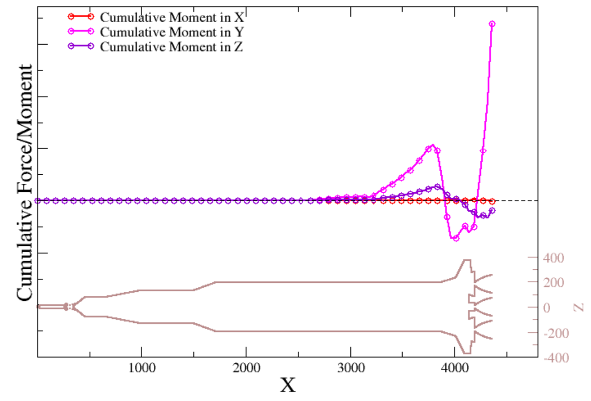


(b) Cumulative.

**Figure 22. The axial drag forces on the Saturn V rocket.**



(a) Lift and side forces.



(b) rolling, pitching, and yawing moments.

**Figure 23. Cumulative forces and moments on the Saturn V rocket.**

The ability to specify a number of bins to control bin size and to cut the triangulation in an arbitrary user-specified direction are shown on an example geometry. The force/moment integration in the bins and the resulting sectional forces and moments are demonstrated to be accurate using a flat plate with exact polynomial and trigonometric functions. A CRM wing and the Saturn V rocket show examples of real life cases.

A user-interface is developed and demonstrated for entering the inputs for the algorithm as well as for plotting the results. The goal of the GUI is to simplify the use of the resulting tool and to provide easy access to the user. The underlying algorithms are designed to be accurate and efficient so that the resulting software can be used to obtain plots of sectional loads from a database of hundreds of CFD solutions in tens of minutes instead of hours.

## V. Acknowledgements

This work was conducted under NASA's Constellation program to support the Ares-I space exploration vehicle. The authors would like to thank Dr. T. Pulliam and J. Onufer for the CRM and Saturn V solutions, respectively. The authors would also like to thank M. Gusman, and Dr. G. Klopfer for providing test cases during the development process. Finally, the authors are grateful to Dr. M. Nemec for providing the box with a wing as well as useful discussions.

## References

[1]Abbott, I. H. and von Doenhoff, A. E., *Theory of Wing Sections*, Dover Publications Inc., 1959.

[2]Lippisch, A., "Method for the Determination of the Spanwise Lift Distribution," NACA-TM-778, Oct. 1935.

[3]Weissinger, J., "The Lift Distribution of Swept-back Wings," NACA-TM-1120, March 1947.

[4]Sivells, J. C., "An Improved Approximate Method for Calculating Lift Distributions due to Twist," NACA-TN-2282, 1951.

[5]Allen, J. M., Shaw, D. S., and Sawyer, W. C., "Analysis of Selected Data from the Triservice Missile Data Base," *Journal of Spacecraft and Rockets*, Vol. 27, No. 1, 1990, pp. 15–20, also AIAA Paper 89–0478, Jan. 1989.

[6]Lesieutre, D., Love, J., and Dillenius, M., "Recent Applications and Improvements to the Engineering Level Aerodynamic Prediction Software MISL3," AIAA-2002-0275, Jan. 2002.

[7]Benek, J. A., Buning, P. G., and Steger, J. L., "A 3-D Chimera Grid Embedding Technique," AIAA Paper 85-1523, June 1985.

[8]Chan, W. M., "Enhancements to the Hybrid Mesh Approach to Surface Loads Integration on Overset Structured Grids," AIAA Paper 2009-3990, June 2009.

[9]Chan, W. M. and Buning, P. G., "User's Manual for FOMOCO Utilities - Force and Moment Computation Tools for Overset Grids," NASA TM 110408, July 1996.

[10]Chan, W. M., "Advances in Software Tools for Pre-processing and Post-processing of Overset Grid Computations," *Proceedings of the 9th International Conference on Numerical Grid Generation in Computational Field Simulations*, San Jose, California, June 2005.

[11]Stambulchik, E., "Grace," [http://plasma-gate.weizmann.ac.il/Grace/], 2000.

[12]Ousterhout, J. K., *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, 1994.

[13]Welch, B. B., *Practical Programming in Tcl and Tk*, Prentice Hall PTR, 1997.

[14]Chan, W. M., "The OVERGRID Interface for Computational Simulations on Overset Grids," AIAA Paper 2002–3188, 2002.

[15]Sclafani, A., Vassberg, J., Rumsey, C., DeHaan, M., and Pulliam, T., "Drag Prediction for the NASA CRM Wing/Body/Tail Using CFL3D and OVERFLOW on an Overset Mesh," AIAA-2010-4219, June 2010.

[16]Nichols, R. and Tramel, R. and Buning, P., "Solver and Turbulence Model Upgrades to OVERFLOW 2 for Unsteady and High-Speed Applications," AIAA Paper 2006–2824, June 2006.